Geometric Differential Evolution for Combinatorial and Programs Spaces

A. Moraglio A.Moraglio@cs.bham.ac.uk
School of Computer Science, University of Birmingham, UK

J. Togelius julian@togelius.com
Center for Computer Games Research, IT University of Copenhagen, Denmark

S. Silva sara@kdbio.inesc-id.pt, sara@dei.uc.pt
INESC-ID, IST Technical University of Lisbon, and CISUC, University of Coimbra, Portugal

Abstract

Geometric differential evolution (GDE) is a recently introduced formal generalization of traditional differential evolution (DE) that can be used to derive specific differential evolution algorithms for both continuous and combinatorial spaces retaining the same geometric interpretation of the dynamics of the DE search across representations. In this article, we first review the theory behind the GDE algorithm, then, we use this framework to formally derive specific GDE for search spaces associated with binary strings, permutations, vectors of permutations and genetic programs. The resulting algorithms are representation-specific differential evolution algorithms searching the target spaces by acting directly on their underlying representations. We present experimental results for each of the new algorithms on a number of well-known problems comprising NK-landscapes, TSP, and Sudoku, for binary strings, permutations and vectors of permutations. We also present results for the Regression, Artificial Ant, Parity and Multiplexer problems within the genetic programming domain. Experiments show that overall the new DE algorithms are competitive with well-tuned standard search algorithms.

Keywords

Differential evolution, representations, principled design of search operators, combinatorial spaces, genetic programming, theory.

Two relatively recent additions to the Evolutionary Algorithms (EAs) family are Particle Swarm Optimization (PSO) [6], inspired to the flocking behavior of swarms of birds, and Differential Evolution (DE) [24], which is similar to PSO, but it uses different equations governing the motion of the particles. In PSO, the velocity and position of each particle (individual)[1] are updated using a linear combination involving the position of the best solution the particle has visited so far and the position of the best particle in the current swarm (population). In DE, the position of each individual is updated using a linear combination of the positions of three individuals picked at random in the current population. Despite their relatedness, DE is known to produce consistently better performance than PSO on many problems. In fact, DE is one of the most competitive EAs for continuous optimization [24] [23].

In their initial inception, both PSO and DE were defined only for continuous problems. In both algorithms, the motion of particles is produced by linear combinations of points in space and has a natural geometric interpretation [19] [10]. There are a number of extensions of DE to binary spaces [23] [22], spaces of permutations [3] [21] and to the space of genetic programs

[1] The position of a particle represents the location of a solution in the search space. Its velocity determines the current search direction from that location and the step size.

[20]. There are also extensions of PSO to binary spaces [5]. Some of these works recast combinatorial optimization problems as continuous optimization problems and then apply the traditional DE algorithm to solve these continuous problems. Other works present DE algorithms defined directly on combinatorial spaces that, however, are only loosely related to the traditional DE in that the original geometric interpretation is lost in the transition from continuous to combinatorial spaces. Furthermore, every time a new solution representation is considered, the DE algorithm needs to be rethought and adapted to the new representation.

GDE [19] is a recently devised formal generalization of DE that, in principle, can be specified to any solution representation while retaining the original geometric interpretation of the dynamics of the points in space of DE across representations. In particular, GDE can be applied to any search space endowed with a distance and associated with any solution representation to derive formally a specific GDE for the target space and for the target representation. GDE is related to Geometric Particle Swarm Optimization (GPSO) [10], which is a formal generalization of the particle swarm optimization algorithm [6]. Specific GPSOs were derived for different types of continuous spaces and for the Hamming space associated with binary strings [11], for spaces associated with permutations [16] and for spaces associated with genetic programs [26].

The present article reviews the theory behind the GDE algorithm, illustrates how this framework can be used in practice as a tool for the principled design of DE search operators for standard and more complex solution representations associated with combinatorial spaces, and finally presents experimental tests and analysis of the new GDE algorithms endowed with such operators on well-known problems. In particular, as target spaces for the GDE, we consider combinatorial spaces associated with binary strings, permutations and vectors of permutations and computer programs represented as expression trees.

The contribution of this article is to show that an existing and very popular algorithm for continuous optimization – Differential Evolution – can be generalized in a mathematically principled way and systematically instantiated to any combinatorial space/representation, endowed with a notion of distance, without introducing any arbitrary element of choice in the transition. So, the derived representation-specific algorithms are in a strong mathematical sense equivalent to the original DE for the continuous space. Whereas the geometric framework makes the instantiation of the DE to new spaces and representations always formally possible, it is important to show that the search operators specified by the theory can be actually constructed in practice. This article shows that it is indeed possible for a number of important, non-trivial representations. From an experimental point of view, the article shows that the new DE algorithms are competitive with standard algorithms on a few well-studied problems. This is a rather interesting achievement, as it is one of the very rare examples in which Evolutionary Computation theory has been able to inform the practice of search operator design successfully.

The practitioner interested in applying the new framework to a specific problem domain on one of the representations presented in this work can skip the theorems and proofs. The article reports pseudo code of the general GDE algorithm and of all representation-specific operators. Also, the parameter space of important parameters is investigated to suggest reasonable parameter settings for each representation.

As for deriving specific GDE for new representations and new distances, it is necessary to have an understanding of the underlying theory. This task suits better the theoretician than the practitioner. However, the long term vision of this line of research is to *automate the design* of DE and other search algorithms for new representations and new problems, so making this theory very practically relevant and useful.

The remaining part of the article is organized as follows. Section 1 contains an introduction to a formal theory of search operators that applies across representations that forms the context for the generalization of the DE algorithm. Section 2 briefly introduces the classic DE algorithm,

and section 3 describes the derivation of the general GDE algorithm. Section 4 presents specific GDE search operators for binary strings, and reports experimental results on NK-landscapes. Section 5 presents specific GDE search operators for permutations, and reports experiments on the Travelling Salesman Problem (TSP). Section 6 presents specific GDE search operators for Sudoku for which candidate solution grids are represented as vectors of permutations, and reports experimental results for this problem. Section 7 presents specific GDE search operators for expression trees, and reports the experimental analysis on standard GP benchmark problems. Section 8 presents conclusions and future work.

1 The Geometry of Representations

In this section, we introduce the ideas behind a recent formal theory of representations [9] which forms the context for the generalization of DE presented in the following sections. See Moraglio's PhD thesis [9], for a complete treatment of this matter.

Search algorithms can be viewed from a geometric perspective [9]. The search space is seen as a geometric space with a notion of distance between points, and candidate solutions are points in the space. For example, search spaces associated with combinatorial optimization problems are commonly represented as graphs in which nodes correspond to candidate solutions and edges between solutions correspond to neighbour candidate solutions. We can endow these spaces with a notion of distance between solutions equal to the length of the shortest path between their corresponding nodes in the graph. Formally, a search space is seen as a metric space.

Definition 1. A metric space is a set X together with a real valued function $d : X \to X$ (called a metric or distance function) such that, for every $x, y, z \in X$:

1. $d(x, y) = 0$ if and only if $x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, z) + d(z, y) \geq d(x, y)$

Geometric search operators are defined using geometric shapes, defined in terms of distance between points in space, to delimit the region of search space where to sample offspring solutions relative to the positions of parent solutions. For example, geometric crossover is a search operator that takes two parent solutions in input corresponding to the end-points of a segment, and returns points sampled at random on the segment as offspring solutions. In order to define formally geometric crossover, we need the notion of metric segment which is a generalization of traditional segment in the Euclidean space to metric spaces.

Definition 2. Let X be a metric space endowed with a metric d. For any $x, y \in X$, the d-metric segment $[x, y]_d$ between x and y is the set $\{z \in X : d(x, z) + d(z, y) = d(x, y)\}$

Geometric crossover is defined formally as follows.

Definition 3. (Geometric crossover [12]) A recombination operator is a geometric crossover under the metric d if for any pair of parents all their offspring are in the d-metric segment between them.

The specific distance associated with the search space at hand is used in the definition of metric segment to determine the specific geometric crossover for that space. Therefore, each search space is associated with a different space-specific geometric crossover. However, all geometric crossovers have the same abstract geometric definition.

Candidate solutions can be seen as points in space, geometric view, or equivalently, as syntactic configurations of a certain type, representation view. For example, a candidate solution in the Hamming space can be considered as a point in space or as a binary string corresponding

to that point. This allows us to think of a search operator equivalently as (i) an algorithmic procedure which manipulates the syntactic configurations of the parent solutions to obtain the syntactic configurations of the offspring solutions using well-defined representation-specific operations (representation/operational view), or (ii) a geometric description which specifies what points in the space can be returned as offspring for the given parent points and with what probability (geometric/abstract view). For example, uniform crossover for binary strings [25] is a recombination operator that produces offspring binary strings by inheriting at each position in the binary string the bit of one parent string or of the other parent string with the same probability. This is an operational view of the uniform crossover that tells how to manipulate the parent strings to obtain the offspring string. Equivalently, the same operator can be defined geometrically as the geometric crossover based on the Hamming distance that takes offspring uniformly at random on the segment between parents [12].

This dual perspective on the geometric search operators has surprising and important consequences [9]. One of them is the possibility of principled generalization of search algorithms from continuous spaces to combinatorial spaces, as sketched in the following.

1. Given a search algorithm defined on continuous spaces, one has to recast the definition of the search operators expressing them explicitly in terms of Euclidean distance between parents and offspring.

2. Then one has to substitute the Euclidean distance with a metric, obtaining a formal search algorithm generalizing the original algorithm based on the continuous space. The formal search algorithm obtained is defined axiomatically as it is based on the axiomatic notion of metric. In particular, it does not refer to any specific instantiation of metric space.

3. Next, one can consider a (discrete) representation and a distance associated with it (a combinatorial space) and use it in the definition of the formal search algorithm to obtain a specific instance of the algorithm for this space.

4. Finally, one can use the geometric description of the search operator to derive its operational definition in terms of manipulation of the specific underlying representation.

This methodology was used to generalize PSO and DE to general metric spaces obtaining the abstract algorithms GPSO [10] and GDE [19] which can then be used as formal specifications to derive specific versions of GPSO and GDE for specific representations and distances. In the following sections, we illustrate how the methodology can be used in practice to generalize DE and to specialize it to specific metric spaces associated with a number of representations. The same methodology can be used to generalize to combinatorial spaces other algorithms naturally based on a notion of distance. This includes search algorithms such as Response Surface Methods, Estimation of Distribution Algorithms and Lipschitz Optimization algorithms, and also Machine Learning algorithms.

2 Classic Differential Evolution

In this section, we describe the traditional DE [2] [23] (see algorithm 1). Note that we consider the objective function f to be maximized.

The characteristic that sets DE apart from other evolutionary algorithms is the presence of the differential mutation operator (see line 5 of algorithm 1). This operator creates a mutant vector U by perturbing a vector $X3$ picked at random from the current population with the scaled difference of other two randomly selected population vectors $F \cdot (X1 - X2)$. The operation

[2] The DE version considered here is known as DE/rand/1/bin, which is perhaps the most well-known. However, many other versions exist [23].

Algorithm 1 DE with differential mutation and discrete recombination

```
1: initialize population of N_p real vectors at random
2: while stop criterion not met do
3:    for all vector X(i) in the population do
4:       pick at random 3 distinct vectors from the current population X1, X2, X3
5:       create mutant vector U = X3 + F · (X1 − X2) where F is the scale factor parameter
6:       set V as the result of the discrete recombination of U and X(i) with probability Cr
7:       if f(V) ≥ f(X(i)) then
8:          set the i^th vector in the next population Y(i) = V
9:       else
10:         set Y(i) = X(i)
11:      end if
12:   end for
13:   for all vector X(i) in the population do
14:      set X(i) = Y(i)
15:   end for
16: end while
```

is understood being important because it adapts the mutation direction and its step size to the level of convergence and spatial distribution of the current population. The mutant vector is then recombined with the currently considered vector $X(i)$ using discrete recombination[3] and the resulting vector V replaces the current vector in the next population if it has better or equal fitness.

The differential mutation parameter F, known as scale factor or amplification factor, is a positive real normally between 0 and 1, but it can take also values greater than 1. The recombination probability parameter Cr of the discrete recombination (algorithm 1, line 6) takes values in $[0, 1]$. It is the probability, for each position in the vector $X(i)$, of the offspring V inheriting the value of the mutant vector U. When $Cr = 1$, the algorithm 1 degenerates to a DE algorithm with differential mutation only (because $V = U$). When $F = 0$, the algorithm 1 degenerates to a DE algorithm with discrete crossover only, as $U = X3$. The population size N_p normally varies from 10 to 100.

3 Geometric Differential Evolution

Following the methodology outlined in section 1, in this section we generalize the classic DE algorithm to general metric spaces. To do it, we recast differential mutation and discrete recombination as functions of the distance of the underlying search space, thereby obtaining their abstract geometric definitions. Then, in the following sections, we derive the specific DE algorithms for binary strings, permutations, vectors of permutations and genetic programs by plugging distances associated with these representations in the abstract geometric definition of the search operators.

3.1 Generalization of differential mutation

Let $X1, X2, X3$ be real vectors and $F \geq 0$ a scalar. The differential mutation operator produces a new vector U as follows:

$$U = X3 + F \cdot (X1 - X2) \tag{1}$$

[3] In order to enforce a modification of $X(i)$, at least one locus of the mutant vector is normally kept during recombination.

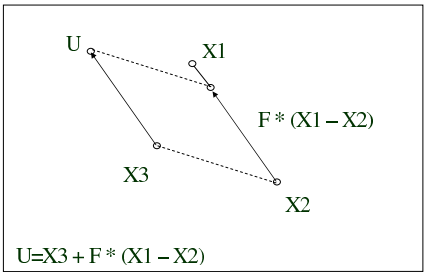

Figure 1: Construction of U using vectors.

The algebraic operations on real vectors in equation 1 can be represented graphically [23] as in figure 1 by means of operations on (graphical) vectors, in which the real vectors $X1, X2, X3$ and U are represented as points.

Unfortunately, the graphical interpretation of equation 1 in terms of operations on vectors does not help us to generalize equation 1 to general metric spaces because the notions of vector and operations on vectors are not well-defined at this level of generality. More formally, a vector space V is a set that is closed under vector addition and scalar multiplication. The basic example is the n-dimensional Euclidean space, where scalars are real numbers. For a general vector space, the scalars are members of a field $\mathbf{F}$. A field is any set of elements endowed with two operations, addition and multiplication, that satisfy a number of axioms, called field axioms. Whereas the n-dimensional Euclidean space can be naturally associated with a notion of distance, via the notion of norm, the underlying set X of a metric space cannot in general be associated with a vector space V over a scalar field $\mathbf{F}$, derived from the metric d of the metric space.

In the following, we propose a generalization based on interpreting equation 1 in terms of segments and extension rays, which are geometric elements well-defined in both vector spaces and metric spaces. To do that, we need the notions of convex combination and extension ray, as follows.

Definition 4. Given vectors $u, v \in V$ a vector space over R^n, and scalars $r, s \in R$, a convex combination is defined as

$$CX(u, v) = r \cdot u + s \cdot v$$

with $r \geq 0$, $s \geq 0$, and $r + s = 1$. The extension ray originating in u and extending beyond v is defined as

$$ER(u, v) = r \cdot u + s \cdot v$$

with $s \geq 0$, and $r + s = 1$.

Note that convex combination and extension ray differ only in the range of the parameter r.

Equation 1 can be rewritten in terms of convex combination as follows:

$$U + F \cdot X2 = X3 + F \cdot X1 \tag{2}$$

By dividing both sides by $1 + F$ and letting $W = \frac{1}{1+F}$ we have:

$$W \cdot U + (1 - W) \cdot X2 = W \cdot X3 + (1 - W) \cdot X1 \tag{3}$$

Both sides of equation 3 are convex combinations of two vectors. On the left-hand side, the vectors U and $X2$ have coefficients W and $1 - W$, respectively. These coefficients sum up to

one and are both positive because $W \in [0, 1]$ for $F \geq 0$[4]. Analogously, the right-hand side is a convex combination of the vectors $X3$ and $X1$ with the same coefficients.

There is an interesting relation between the algebraic notion of convex combination of two vectors and the geometric notion of segment in the Euclidean space. Vectors represent points in space. Any point P_C corresponding to a vector C obtained by a convex combination of two vectors A and B lay in the line segment between their corresponding points P_A and P_B. The vice versa also holds true: any vector C corresponding to a point P_C of the segment $[P_A, P_B]$ can be obtained as a convex combinations of the vectors A and B. For each point on the segment, the weights W_A and W_B in the convex combination localize the point P_C on the segment $[P_A, P_B]$: distances to P_C from P_A and P_B are inversely proportional to the corresponding weights, W_A and W_B ,i.e., $d(P_A, P_C) \cdot W_A = d(P_B, P_C) \cdot W_B$.

The above relation allows for a geometric interpretation of equation 3 in terms of convex combinations (see figure 2). Let us call E the vector obtained by the convex combinations on both sides of equation 3. Geometrically the point E must be the intersection point of the segments $[U, X2]$ and $[X1, X3]$. The distances from E to the endpoints of these segments can be determined from equation 3 as they are inversely proportional to their respective weights (i.e., $d(U, E) \cdot W = d(X2, E) \cdot (1 - W)$ and $d(X1, E) \cdot (1 - W) = d(X3, E) \cdot W$). Since the point U is unknown (but its weight is known), it can be determined geometrically by firstly determining $E = CX(X1, X3)$, the convex combination of $X1$ and $X3$; then, by projecting $X2$ beyond E obtaining a point $U = ER(X2, E)$, i.e., the extension ray originating in $X2$ and passing through E, such that the proportions of the distances of $X2$ and U to the point E is inversely proportional to their weights. In the Euclidean space, the constructions of U using vectors (figure 1) and convex combinations (figure 2) are equivalent (algebraically, hence geometrically).

Segments and extension rays in the Euclidean space and their weighted extensions can be expressed in terms of distances, hence, these geometric objects can be naturally generalized to metric spaces by replacing the Euclidean distance with a metric. We will present their abstract definitions in section 3.3.

The differential mutation operator $U = DM(X1, X2, X3)$ with scale factor F can now be defined for any metric space following the construction of U presented in figure 2 as follows:

1. Compute $W = \frac{1}{1+F}$
2. Get E as the convex combination $CX(X1, X3)$ with weights $(1 - W, W)$ (generalizing $E = (1 - W) \cdot X1 + W \cdot X3$)
3. Get U as the extension ray $ER(X2, E)$ with weights $(W, 1 - W)$ (generalizing $U = (E - (1 - W) \cdot X2)/W$)

3.2 Generalization of discrete recombination

After applying differential mutation, the DE algorithm applies discrete recombination to U and $X(i)$ generating V. Discrete recombination is a geometric crossover under Hamming distance for real vectors[5] [9]. The Hamming distance (HD) for real vectors is defined analogously to the Hamming distance between binary strings: it is the number of sites with mismatching values across the two vectors. This distance is provably a metric as it is a product metric of the discrete

[4] As F is generally in the range $[0, 1]$, the corresponding range for W is in fact only $[0.5, 1]$. The value of F should be chosen larger than one, may larger W be needed in different spaces.

[5] The name Hamming distance for real vectors derives from the observation that when the domain of real vectors is restricted to the set of binary strings, the distance on the restricted domain coincides with the traditional notion of Hamming distance for binary strings.

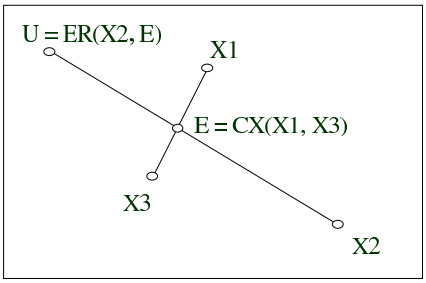

Figure 2: Construction of U using convex combination and extension ray.

metric on R [9]. From its definition, we can derive that the Cr parameter of the discrete recombination is proportional to the expected number of values that V inherits from U. Therefore, $E[HD(U,V)] = Cr \cdot HD(U,X(i))$ and $E[HD(X(i),V)] = (1-Cr) \cdot HD(U,X(i))$. Consequently, Cr and $1-Cr$ can be interpreted as the weights of U and $X(i)$, respectively, of the convex combination that returns V in the space of real vectors endowed with Hamming distance. In order to generalize the discrete recombination, by replacing Hamming distance with a metric, we obtain the abstract convex combination operator CX introduced in the previous section. So, we have that the generalized discrete recombination of U and $X(i)$ with probability parameter Cr generating V is as follows: $V = CX(U, X(i))$ with weights $(Cr, 1-Cr)$.

In the classic DE (algorithm 1), replacing the original differential mutation and discrete recombination operators with their generalizations, we obtain the formal Geometric Differential Evolution (see algorithm 2). When the formal algorithm is specified on the Euclidean space, the resulting Euclidean GDE does *not* coincide with the classic DE. This is because, whereas the original differential mutation operator can be expressed as a function of the Euclidean distance, the original discrete recombination operator can be expressed as a function of the Hamming distance for real vectors, not of the Euclidean distance. The Euclidean GDE coincides with an existing variant of traditional DE [23], which has the same differential mutation operator but in which the discrete recombination is replaced with blend crossover. Interestingly, blend crossover lives in the same space as differential mutation and their joint behavior has a geometric interpretation in space.

3.3 Definition of convex combination and extension ray

A notion of convex combination in metric spaces was introduced in the GPSO framework [10]. The notion of extension ray in metric spaces was introduced in the GDE framework [19]. In the following, we review their definitions and emphasize that the extended ray recombination can be naturally interpreted as the inverse operation of the convex combination.

In section 1, we introduced the notion of metric segment. Let us recall the definition of extension ray in metric spaces. The extension ray $ER(A,B)$ in the Euclidean plane is a semi-line originating in A and passing through B (note that $ER(A,B) \neq ER(B,A)$). The extension ray in a metric space can be defined indirectly using metric segments, as follows.

Definition 5. Given points A and B, the metric extension ray $ER(A,B)$ is the set of points which comprises any point C that satisfies $C \in [A,B]$ or $B \in [A,C]$.

Only the part of the extension ray beyond B will be of interest because any point C that we want to determine, which is, the offspring of the differential mutation operator, is never between A and B by construction.

We can now use these geometric objects as basis for defining the convex combination operator and the extended ray recombination operator in *metric spaces*, as follows.

Algorithm 2 Formal Geometric Differential Evolution

initialize population of N_p configurations at random
while stop criterion not met **do**
  for all configuration $X(i)$ in the population **do**
    pick at random 3 distinct configurations from the current population $X1, X2, X3$
    set $W = \frac{1}{1+F}$ where F is the scale factor parameter
    create intermediate configuration E as the convex combination $CX(X1, X3)$ with weights $(1 - W, W)$
    create mutant configuration U as the extension ray $ER(X2, E)$ with weights $(W, 1 - W)$
    create candidate configuration V as the convex combination $CX(U, X(i))$ with weights $(Cr, 1 - Cr)$ where Cr is the recombination parameter
    if $f(V) \geq f(X(i))$ **then**
      set the i^{th} configuration in the next population $Y(i) = V$
    else
      set $Y(i) = X(i)$
    end if
  end for
  for all configuration $X(i)$ in the population **do**
    set $X(i) = Y(i)$
  end for
end while

Definition 6. Let X be a metric space endowed with distance function d. The *convex combination in metric spaces* $CX((A, W_A), (B, W_B))$ of two points A and B in X with weights W_A and W_B (positive and summing up to one) returns the set of points comprising any point C in X such that $C \in [A, B]$ and $d(A, C) \cdot W_A = d(B, C) \cdot W_B$.

When specified to Euclidean spaces, this notion of convex combination coincides with the traditional notion of convex combination of real vectors. Notice that, unlike the Euclidean case, in other spaces, for specific points A and B and specific choices of W_A and W_B, the convex combination in metric spaces may return a set that contains a single point, the empty set, or a set containing more than a point.

The *extension ray recombination in metric spaces* ER is defined as the inverse operation of the weighted convex combination CX, as follows.

Definition 7. Let X be a metric space endowed with distance function d. The weighted extension ray $ER((A, W_{ab}), (B, W_{bc}))$ of the points A (origin) and B (through) and weights W_{ab} and W_{bc} returns a set of points comprising any point C such that the set of points returned by the convex combination of C with A with weights W_{bc} and W_{ab}, i.e., $CX((A, W_{ab}), (C, W_{bc}))$, includes point B.

Notice that from the above definition follows that the weights W_{ab} and W_{bc} in ER are positive real numbers between 0 and 1 and sum up to 1 because they must respect this condition in $CX((A, W_{ab}), (C, W_{bc}))$. The set of points returned by the weighted extension ray in metric spaces ER can be characterized explicitly in terms of distances to the input points of ER, as follows [19].

Lemma 1. *The points returned by the weighted extension ray* $ER((A, W_{ab}), (B, W_{bc}))$ *comprises any point* C *which is at a distance* $d(A, B) \cdot W_{ab}/W_{bc}$ *from* B and *at a distance* $d(A, B)/W_{bc}$ *from* A.

Proof. From the definition of weighted extension ray we have that $B \in CX((A, W_{ab}), (C, W_{bc}))$. Hence, $d(A, C) = d(A, B) + d(B, C)$ and the distances $d(A, B)$ and $d(B, C)$ are inversely proportional to the weights W_{ab} and W_{bc}, i.e., $d(A, B) \cdot W_{ab} = d(B, C) \cdot W_{bc}$. Consequently, $d(A, C) = d(A, B)/W_{bc}$ and substituting it in $d(B, C) = d(A, C) - d(A, B)$ we get $d(B, C) = d(A, B) \cdot W_{ab}/W_{bc}$, since $W_{ab} + W_{bc} = 1$. □

This characterization is useful to construct procedures to implement the weighted extension ray for specific spaces. In fact, we used it, together with representation-specific properties of the extension ray, in the derivation of the extension ray recombination operators for all representations in this article.

Importantly, the above definitions of convex combination and extension ray in metric spaces can be made stochastic and relaxed treating their output points as the outcomes of a random variable which are required to meet the relation between weights and distances only in expectation. More precisely, the convex combination $CX((A, W_A), (B, W_B))$ is understood as a stochastic operator whose output is a random variable C for which it must hold that (i) the support set of C is included in $[A, B]$ and (ii) $E[d(A, C)] \cdot W_A = E[d(B, C)] \cdot W_B$. An analogous stochastic definition can be made for the extension ray. *These relaxed versions of the operators have the advantage of being more naturally suited to combinatorial spaces* and being easier to implement for such spaces.

4 Binary GDE

In this section, we derive formally specific convex combination and extension ray recombination for the Hamming space for binary strings. These specific operators can then be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the Hamming space, the Binary GDE.

4.1 Convex combination

Let us consider the convex combination $C = CX((A, W_A), (B, W_B))$ of two points A and B with weights W_A and W_B (positive and summing up to one). In the Euclidean space, C is uniquely determined, however this is not the case for all metric spaces. In particular, it does not hold for Hamming spaces. When CX is specified to Hamming spaces on binary strings, it can be formally shown that we obtain the recombination operator outlined in algorithm 3 [10]. This algorithm returns an offspring binary string C of parent binary strings A and B, where C is interpreted as a random variable on $[A, B]$, such that $E[HD(A, C)]/E[HD(B, C)] = W_B/W_A$ (i.e., the relation holds in expectation as defined in the previous section), where HD denotes the Hamming distance between binary strings. This differs from the Euclidean case where the ratio is guaranteed.

4.2 Extension ray

In order to gain an intuitive understanding of how an extension ray looks like in the Hamming space, let us consider an example of extension ray originating in $A = 110011$ and passing through $B = 111001$.

The relation $C \in [A, B]$ is satisfied by those C that match the schema $S1 = 11 * 0 * 1$. This is the set of the possible offspring of A and B that can be obtained by recombining them using the uniform crossover.

The relation $B \in [A, C]$ is satisfied by all those C that match $S2 = * * 1 * 0*$. This is the set of all those C that when recombined with A using the uniform crossover can produce B as offspring.

The following theorem characterizes the extension ray in the Hamming space in terms of schemata.

Algorithm 3 Binary Convex Combination Operator

1: inputs: binary strings A and B and weights W_A and W_B (weights must be positive and sum up to 1)
2: **for all** position i in the strings **do**
3: **if** random(0,1) $\leq W_A$ **then**
4: set $C(i)$ to $A(i)$
5: **else**
6: set $C(i)$ to $B(i)$
7: **end if**
8: **end for**
9: return string C as offspring

Theorem 2. *Let A and B be fixed binary strings in the Hamming space:*

1. *the relation $C \in [A, B]$ is satisfied by those strings C that match the schema obtained by keeping the common bits in A and B and inserting $*$ where the bits of A and B do not match.*

2. *the relation $B \in [A, C]$ is satisfied by all those strings C that match the schema obtained by inserting $*$ where the bits are common in A and B and inserting the bits coming from B where the bits of A and B do not match.*

Proof. Proof of statement 1: the schema so defined corresponds to the set of the possible offspring of A and B that can be obtained by recombining them using the uniform crossover. This crossover operator corresponds to the uniform geometric crossover under Hamming distance which returns offspring on the segment between parents.

Proof of statement 2: all C matching the schema S defined in the second statement recombined with A can produce B as offspring. This is because, at positions where the schema S presents $*$, the bit in B can be inherited from A. At positions where the schema S has 0 or 1, the bit in B can be inherited from C. Furthermore, only the strings C matching S can produce B when C is recombined with A. □

Using the characterization of the weighted extension ray in terms of distances (lemma 1) and the characterization of the extension ray in the Hamming space in terms of schemata (theorem 2), we were able to derive the weighted extension ray recombination for this space (see algorithm 4). Theorem 3 proves that recombination operator conforms to the definition of weighted extension ray for the Hamming space (in expectation).

Theorem 3. *Given parents A and B, the stochastic recombination in algorithm 4 returns random offspring C such that $Pr(B \in [A, C]) = 1$ and $E[HD(B, C)]/HD(A, B) = W_{AB}/W_{BC}$, where $E[HD(B, C)]$ is the expected Hamming distance between B and the offspring C (with respect to the probability distribution of C).*

Proof. This can be shown as follows. The number of bits in which A and B differ are $HD(A, B)$. The number of bits in which A and B do not differ is $n - HD(A, B)$. For the bits in which A and B differ, the string C equals B. For each bit in which A and B do not differ, C does not equal B with probability p. So, the expected distance between B and C is $E[HD(B, C)] = (n - HD(A, B)) \cdot p$. By substituting $p = HD(B, C)/(n - HD(A, B))$, we have $E[HD(B, C)] = HD(B, C) = HD(A, B) \cdot W_{AB}/W_{BC}$. So, $E[HD(B, C)]/HD(A, B) = W_{AB}/W_{BC}$. □

Algorithm 4 Binary Extension Ray Recombination

1: inputs: binary strings A (origin) and B (through) of length n and weights W_{AB} and W_{BC} (weights must be positive and sum up to 1)
2: set $HD(A,B)$ as Hamming distance between A and B
3: set $HD(B,C)$ as $HD(A,B) \cdot W_{AB}/W_{BC}$ rounded to the closest integer (estimate the distance between B and C using the weights)
4: set p as $HD(B,C)/(n - HD(A,B))$ (this is the probability of flipping bits away from A and B beyond B)
5: **for all** position i in the strings **do**
6: set $C(i) = B(i)$
7: **if** $B(i) = A(i)$ and random(0,1) $\leq p$ **then**
8: set $C(i)$ to the complement of $B(i)$
9: **end if**
10: **end for**
11: return string C as offspring

Theorem 3 holds under the assumption that the diameter of the space is at least as large as the wanted Hamming distance between A and C. That is, that the requested point on the extension ray does not go beyond the boundary of the space. When such a condition does not hold, the value of p becomes lager than 1, and the offspring C returned by the algorithm 4 is the point on the extension ray at maximum distance from A, i.e., the same offspring which is returned when $p = 1$. In this case, the required relation between distance and weights does not hold.

Now we have operational definitions of convex combination and extension ray for the space of binary strings under HD. These space-specific operators can be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the space of binary strings.

4.3 Experiments for Binary GDE

We implemented the GDE algorithm for binary spaces within a Java framework[6]. In order to systematically test the behaviour of GDE on landscapes with varying amount of epistasis, we performed experiments using NK fitness landscapes, as proposed by Kauffman [4]. NK landscapes have two parameters: N, the number of dimensions, was fixed to 100 in our experiments; K, the number of dependencies on other loci per locus was varied between 0 and 5.

The proposed algorithm was compared with three other algorithms:

- cGA: A canonical Genetic Algorithm, with roulette wheel fitness-proportionate selection, uniform crossover and bitflip mutation.
- tGA: A Genetic Algorithm with truncation selection, with a selection threshold of $popsize/2$.
- ES: A $\mu + \lambda$ Evolution Strategy, with $\mu = \lambda = popsize/2$ and bitflip mutation.

These are basic standard evolutionary algorithms and were chosen to have simple and well-known methods to compare against. Two types of Genetic Algorithms differing in the selection scheme used are considered because we found that the selection scheme may affect the performance significantly.

[6] Source code is available upon request from the second author.

For the ES and GAs, the bitflip mutation works as follows: each bit in the chromosome is considered, and with probability p this bit is flipped. In the experiments involving these algorithms, the parameter p was systematically varied between 0.0 and 0.5 in increments of 0.01. For the experiments involving GDE, the key parameters F and Cr were systematically varied between 0.0 and 1.0 in increments of 0.1.

All evolutionary runs lasted for 10000 function evaluations, which were allocated either as population size 100 and 100 generations or as population size 10 and 1000 generations. The CPU time of the different algorithms was very similar in all cases.

The results in Table 1 show that GDE is a very competitive algorithm overall on this problem. For population size 100, GDE is the best of the four algorithms for $K > 1$, and for population size 10, GDE is the best-performing algorithm when $0 < K < 4$. The difference is statistically significant ($p < 0.001$ using a two-tailed student's t-test) for $K = 3$ (both population sizes) and $K = 1$ (population size 10). GDE is never much worse than the best algorithm, even for those values of K where it is not the best algorithm.

Table 2 shows the best parameter settings for GDE for different K. Apparently, for low K larger population sizes are preferred, and for higher K smaller populations do better. Interestingly, for all K the best configuration is very low F and medium to high Cr. Table 3 presents the best mutation settings found for ES and GA. The GAs always performed best with population size 100, and the ES with population size 10. A clear trend is that ES works best with small populations and both GAs with larger populations; ES also generally prefers lower mutation rates than the GAs.

10	$K = 0$	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$
GDE	0.675 (0.0)	0.685 (0.019)	0.741 (0.02)	0.768 (0.063)	0.752 (0.102)	0.733 (0.136)
cGA	0.579 (0.098)	0.620 (0.146)	0.598 (0.117)	0.613 (0.130)	0.603 (0.127)	0.607 (0.116)
tGA	0.651 (0.011)	0.709 (0.0419)	0.736 (0.045)	0.731 (0.079)	0.747 (0.092)	0.725 (0.11)
ES	0.679 (0.02)	0.698 (0.036)	0.72 (0.063)	0.717 (0.071)	0.720 (0.099)	0.722 (0.104)
100	$K = 0$	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$
GDE	0.649 (0.01)	0.722 (0.049)	0.695 (0.081)	0.715 (0.09)	0.689 (0.113)	0.683 (0.14)
cGA	0.495 (0.138)	0.511 (0.183)	0.528 (0.191)	0.526 (0.217)	0.528 (0.189)	0.517 (0.198)
tGA	0.587 (0.113)	0.605 (0.119)	0.620 (0.133)	0.624 (0.119)	0.629 (0.156)	0.628 (0.121)
ES	0.657 (0.035)	0.693 (0.069)	0.681 (0.097)	0.673 (0.089)	0.692 (0.096)	0.685 (0.109)

Table 1: Results on the NK landscape benchmark. Average maximum fitness at the last generation (standard deviations in parentheses) for each algorithm using K values between 0 and 5, using population sizes of both 10 and 100. 50 runs were performed for each configuration.

K	pop/gen	F	Cr
0	100/100	0.0	0.8
1	100/100	0.0	0.7
2	100/100	0.0	0.5
3	10/1000	0.1	0.9
4	10/1000	0.1	0.8
5	10/1000	0.1	0.8

Table 2: Best parameter settings found for GDE on the NK landscape benchmark.

K	cGA	tGA	ES
0	0.01	0.35	0.01
1	0.01	0.43	0.03
2	0.28	0.47	0.03
3	0.16	0.19	0.04
4	0.39	0.36	0.02
5	0.20	0.30	0.02

Table 3: Best mutation settings for GAs and ES on the NK landscape benchmark.

4.3.1 Discussion

We have found GDE to be competitive with the best of the tested algorithms. For NK landscapes, GDE performs overall best of the standard algorithms we tested it against given roughly the same

amount of automatic parameter tuning.

The best parameters found for GDE show that high Cr and low F values seem to be the best choice in most cases. However, although we have not included a table showing this, the algorithm is not very sensitive to parameters; in particular, F can be varied within the low range on NK landscapes with very little performance difference. The exception is that extreme values of Cr result in drastic performance drops.

5 Permutation-based GDE

In this section, we derive formally specific convex combination and extension ray recombination for the space of permutations. We use the swap distance between permutations as basis for the GDE. These specific operators can then be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the space of permutations, the permutation-based GDE. Notice, however, that in principle, we could choose any other distance between permutations (e.g., adjacent swap distance, reversal distance, insertion distance, etc.) as a basis of the GDE. In that case, for each distance, we would obtain a different permutation-based GDE.

5.1 Swap distance

The swap distance $SD(A, B)$ between two permutations A and B is the minimum number of swaps needed to order one permutation into the order of the other permutation. It can be implemented by counting the number of swaps employed by the selection sort algorithm as in Algorithm 5, which is provably minimal.

Algorithm 5 Swap distance

1: inputs: permutations p_a and p_b
2: set $dist = 0$
3: **for all** position i in the permutations **do**
4: **if** $p_a(i) \neq p_b(i)$ **then**
5: find $p_a(i)$ in p_b and be j its position in p_b
6: swap contents of $p_b(i)$ and $p_b(j)$
7: $dist = dist + 1$
8: **end if**
9: **end for**
10: return $dist$

5.2 Convex combination

Algorithm 6 presents a recombination operator for permutations that was introduced in the GPSO framework [11]. This operator produces an offspring by sorting by swaps the two parent permutations one towards the other until they converge to the same permutation. A random recombination mask of the size of the parent permutations is generated using the parents weights interpreted as probabilities and tossing multiple times a biased coin to select which parents to consider at each position in the mask. The outcome at a specific position dictates which of the two permutations has to be sorted toward the other at that position. This operator was proved to be a convex combination for permutations under swap distance SD in [11]. We report only the statement of the theorem and refer the interested reader to that reference.

Theorem 4. *The convex combination in Algorithm 6 is a geometric crossover under swap distance.*

Additionally, in previous work [11], it was shown that the distances of the parents to the offspring are decreasing functions of their weights in the convex combination. In the following,

Algorithm 6 Convex combination

```
inputs: permutations p_a and p_b, and their weights W_a and W_b
generate a recombination mask m randomly with 'a' and 'b' with probabilities W_a and W_b
for all position i in the permutations do
  if p_a(i) ≠ p_b(i) then
    if m(i) = a then
      find p_a(i) in p_b and be j its position in p_b
      swap contents of p_b(i) and p_b(j)
    else
      find p_b(i) in p_a and be j its position in p_a
      swap contents of p_a(i) and p_a(j)
    end if
  end if
end for
return p_a as offspring
```

we give a stronger result that says that these distances are inversely proportional to the corresponding weights, as required by the refined definition of convex combination introduced in this article.

Theorem 5. *The stochastic convex combination in Algorithm 6 is in expectation a convex combination in the space of permutations endowed with swap distance.*

Proof. The convex combination for permutations is a geometric crossover under swap distance. Therefore, the offspring of the convex combination are on the segment between parents as required to be a convex combination. To complete the proof, we need to show that the weights W_a and W_b of the convex combination are inversely proportional to the expected distances $E[SD(p_a, p_c)]$, $E[SD(p_b, p_c)]$ from the parents p_a and p_b to their offspring p_c, i.e., $E[SD(p_a, p_c)] \cdot W_a = E[SD(p_b, p_c)] \cdot W_b$, as follows.

The recombination mask m contains a set of independently generated choices. Let us consider a generic position, that we call current position. When p_a and p_b differ at the current position, the effect of the corresponding choice in the mask m is sorting p_a a single swap towards p_b with probability W_b and sorting p_b a single swap towards p_a with probability W_a. When p_a and p_b are equal at the current position, the effect of the choice is to leave p_a and p_b unchanged. When all choices in the mask m have been applied p_a and p_b have become equal in all positions, hence converged to the offspring p_c. Since the convex combination operator is a geometric crossover, the offspring p_c is on a shortest path between p_a and p_b (shortest sorting trajectory by swaps). The expected number of swap moves on the shortest path from p_a toward p_b to reach p_c, i.e., $E[SD(p_a, p_c)]$, is given by the number of swap moves on the shortest path, i.e., $SD(p_a, p_b)$, multiplied by the probability that any swap move on the shortest path was obtained by ordering p_a toward p_b, i.e., W_b. Hence $E[SD(p_a, p_c)] = SD(p_a, p_b) \cdot W_b$. Analogously for the other parent we obtain: $E[SD(p_b, p_c)] = SD(p_a, p_b) \cdot W_a$. Therefore, the expected distances of the parents to the offspring are inversely proportional to their respective weights. □

5.3 Extension ray

Algorithm 7 presents a recombination operator that is allegedly the extension ray recombination for permutations under swap distance. This operator produces an offspring permutation by sorting by swaps parent permutation p_b away from parent permutation p_a. The number of swaps away is calculated in a way to obtain consistency between weights and distances of the offspring

to the parents as required from the general definition of extension ray recombination in metric space. The following theorem proves that this is indeed an extension ray recombination for permutations under swap distance.

Algorithm 7 Extension ray recombination

inputs: parent p_a (origin point of the ray) and p_b (passing through point of the ray), with corresponding weights W_{ab} and W_{bc} (both weights are between 0 and 1 and sum up to 1)
output: a single offspring p_c (a point on the extension ray beyond p_b on the ray originating in p_a and passing through p_b)
compute the swap distance $SD(p_a, p_b)$ between p_a and p_b
set $SD(p_b, p_c) = SD(p_a, p_b) \cdot W_{ab}/W_{bc}$ (compute the distance between p_b and p_c using the weights)
set $p = SD(p_b, p_c)/(n - 1 - SD(p_a, p_b))$ (the probability p of swapping elements away from p_a and p_b beyond p_b)
set $p_c = p_b$
for all position i in the permutations **do**
  if $p_c(i) = p_a(i)$ and random(0,1) $\leq p$ **then**
    select at random a position j
    swap contents of $p_c(i)$ and $p_c(j)$
  end if
end for
return p_c as offspring

Theorem 6. *The stochastic extension ray recombination in Algorithm 7 is in expectation an extension ray operator in the space of permutations endowed with swap distance.*

Proof. First we prove that $p_c = ER(p_a, p_b)$ by proving that p_b is on the segment between p_a and p_c under swap distance. Then we prove that the expected distances $E[SD(p_a, p_b)]$ and $E[SD(p_b, p_c)]$ are inversely proportional to the weights W_{ab} and W_{bc}, respectively (i.e., $E[SD(p_a, p_b)] \cdot W_{ab} = E[SD(p_b, p_c)] \cdot W_{bc}$).

Every swap move applied to p_b that increases the Hamming distance between p_a and p_b generates a permutation p'_b such that p_b is on a swap shortest path between p_a and p'_b. This is because (i) p'_b is a swap away from p_b, i.e., $SD(p_b, p'_b) = 1$ and (ii) p'_b is a swap further away from p_a since $HD(p_a, p'_b) > HD(p_a, p_b)$, i.e., $SD(p_a, p_b) + 1 = SD(p_a, p'_b)$. Hence $SD(p_a, p_b) + SD(p_b, p'_b) = SD(p_a, p'_b)$. This construction can be continued applying a swap move to p'_b obtaining a p''_b such that p'_b and p_b are on a swap shortest path between p_a and p''_b. Analogously, for any further reiteration, we obtain $p_b^{(n)}$ such that p_b is on a swap shortest path between p_a and $p_b^{(n)}$. Since the operator ER constructs the offspring p_c (corresponding to $p_b^{(n)}$) from parents p_a and p_b following the above procedure, we have that p_b is on the segment between p_a and p_c under swap distance.

The probability p is the probability of applying a swap away from p_a for each position i, for which p_a equals p_b. Hence, the probability p together with the number k of positions at which p_a equals p_b determine the expected number of swaps away p_c is from p_b. Therefore, $E[SD(p_b, p_c)] = p \cdot k$. For the theorem to hold, the probability p must determine the distance $E[SD(p_b, p_c)]$ such that distances and weights of parents are inversely proportional i.e., $E[SD(p_a, p_b)] \cdot W_{ab} = E[SD(p_b, p_c)] \cdot W_{bc}$. So the required p for this to happen is $p = E[SD(p_b, p_c)]/k$ where $E[SD(p_b, p_c)] = E[SD(p_a, p_b)] \cdot W_{ab}/W_{bc}$. The number k is well-estimated by the length of the diameter of the space (maximum swap distance between any

two permutations), which is $n-1$ where n is the number of elements in the permutation, minus the swap distance between p_a and p_b, i.e., $k \approx (n-1-E[SD(p_a,p_b)])$. For this value of k, the probability p is $E[SD(p_b,p_c)]/(n-1-E[SD(p_a,p_b)])$, which is the one in algorithm 7. Hence, the theorem holds. □

As for the case of the Hamming space, the extension ray recombination operator for permutations cannot return points which are farther away than the diameter of the space. When input weights require this, the point actually returned by the operator is the farthest away point on the extension ray.

Now we have operational definitions of convex combination and extension ray for the space of permutations under swap distance. These space-specific operators can be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the space of permutations.

5.4 Experiments with GDE on TSP

We have tested the permutation-based GDE on randomly generated instances of the Travelling Salesman Problem (TSP), which is perhaps the most famous permutation-based optimization problem. We use TSP as a benchmark problem on which to study the performance of GDE under various parameters, and on which to compare it against two standard search algorithms; we do not try to beat the best customised TSP algorithms.

While we have not tailored our search operators particularly to TSP, it is worth noting that the neighborhood structure on the TSP that works best with local search heuristics is that based on the 2-opt move which reverses the order of the elements in a continuous section of the permutation. Analogously to the swap move, the 2-opt move gives rise to a distance between permutations (known as reversal distance). This would be perhaps the most suitable distance to use as a base for GDE when applied to TSP, as a geometric crossover based on the reversal distance [18] beats the Edge Recombination Crossover [27] which is a crossover specifically tailored to the TSP that performs extremely well. However, computing the reversal distance is NP-Hard [1], and only an approximation of the geometric crossover associated with this distance can be implemented efficiently. Furthermore, the approximated geometric crossover is rather complex. As the swap move gives rise to simple geometric operators, it is a better choice for an initial exploration of the GDE framework. We will derive and test a GDE based on reversal distance in future work.

Local search heuristics based on the swap move are known to do reasonably well on the TSP. Also, genetic algorithms with the PMX crossover operator for permutation, which is known to be a geometric crossover under swap distance, does reasonably well on the TSP. Therefore, as a reference, we compare the GDE on the swap space with a stochastic hill-climber based on the swap move and with a genetic algorithm with rank-based selection, PMX crossover and swap mutation.

The TSP instances used in our experiments are randomly generated, with either 10, 20, 50, 100 or 200 cities. The distance between each pair of cities lies between 0 and 1, and the instances are symmetric but not Euclidean. Twenty TSP instances were generated at the beginning of the experiments; every algorithm configuration is run once per instance, and the fitness averaged over all instances.

Moderately extensive tuning experiments were performed for the population-based algorithms. All algorithms (GDE, GA and hill climber) were run for 100000 (hundred thousand) function evaluations. For GDE and GA, population sizes of 10, 20, 50, 100, 1000 were tried, with the number of generations set to $100000/popsize$. For both algorithms, their two respective key parameters were varied between 0 and 1 in increments of 0.2; for GDE, the parameters are F and Cr. For the GA, these parameters were defined as the elite proportion (how large part of the rank-ordered population is used as the elite; the lesser fit rest of the population is re-

placed each generation) and mutation probability (the probability that a new offspring is created through swap mutation from the previous individual at the same position in the population rather than using PMX crossover of two randomly selected individuals in the population). We note that some extreme settings yield degenerate versions of both algorithms. Alas, for the hillclimber, there is nothing to tune.

Algorithm	Fitness	Population	Parameters
Tour length 10			
Hillclimber	1.9 (0.422)	-	-
GA	1.532 (0.408)	50	elite 0.0, mut 0.4
GDE	1.407 (0.375)	1000	F 0.4, Cr 0.4
Tour length 20			
Hillclimber	2.871 (0.558)	-	-
GA	2.30 (0.388)	1000	elite 0.2, mut 0.2
GDE	2.167 (0.37)	10	F 0.0, Cr 0.2
Tour length 50			
Hillclimber	5.457 (0.609)	-	-
GA	5.399 (0.638)	10	elite 0.2, mut 0.6
GDE	5.24 (0.429)	10	F 0.0, Cr 0.2
Tour length 100			
Hillclimber	8.758 (0.758)	-	-
GA	9.945 (0.782)	10	elite 0.2, mut 0.4
GDE	15.899 (1.707)	10	F 0.1, Cr 0.4
Tour length 200			
Hillclimber	15.099 (0.794)	-	-
GA	21.92	10	elite 0.2, mut 0.4
GDE	48.926 (1.173)	10	F 0.2, Cr 0.2

Table 4: Results on TSP instances of sizes 10, 20, 50, 100 and 200. Standard deviations in parentheses. All fitnesses and standard deviations are calculated over 100 independent runs. Lower fitnesses are better.

The results can be found in table 4. Generally, on small instance sizes (tour lengths of less than 100) a well-tuned GDE is clearly competitive with a well-tuned GA. For tour lengths 10 and 20, the GDE is significantly better ($p < 0.05$ with two-tailed student's t-test). On larger instances, GDE loses out to the GA by a large margin. However, on these instances the GA also performs worse than a standard hillclimber.

For the smallest instances, large populations seemed to work best with GDE; for all the other instances, population size 10 performed best. The best values for both F and Cr were typically low.

Figure 3 dives deeper into parameter space, presenting a comparison of the settings of the F and Cr parameters on TSP instances of size 10, 50 and 200 respectively. For short tours, GDE is not very sensitive to parameters and performs well under most settings, with the exceptions of $Cr = 0$ and ($Cr = 1$ when $F = 0$ or $F = 0.2$). Generally, the best setting for both parameters is low both but not 0; setting both F and Cr to 0.2 yields the best or second best setting for all three instance sizes.

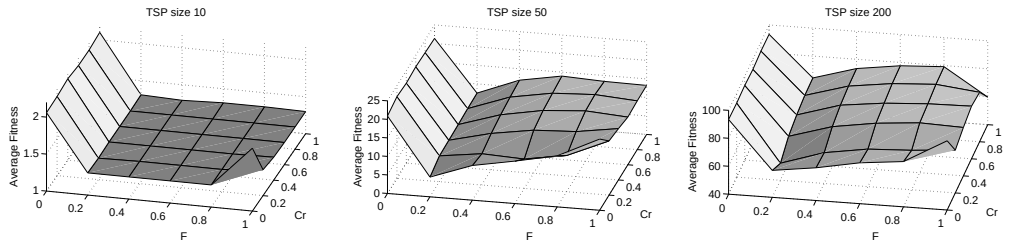

Figure 3: GDE parameters for TSP tours of lengths 10, 50 and 200, and corresponding fitness on the y axis. Lower fitnesses are better. The best-performing population sizes were used for each tour length, meaning size 1000 for length 10 and size 10 for lengths 50 and 200.

5.4.1 Analyzing the failure of GDE on long TSP tours

In the attempt to investigate the reason of the failure of the algorithm for larger tour lengths, we experimentally analysed the dynamics of population fitness and average distance of individuals in the population varying the parameters Cr and F systematically from 0 to 1 with increments of 0.2 for a total of 36 combinations. We studied populations of size 10 for TSP tours of length 100. A short summary of the analysis (table not reported) is as follows. The algorithm presents a variety of different behaviours. For $F = 0$, there is always premature convergence, and for $Cr = 0$ there is little useful progress. $Cr = 0.2$ and $Cr = 0.4$ for almost all values of F tend to be the more stable configurations of the algorithm, which produces steady improvements without loss of diversity, but the progress is very slow and eventually halts without convergence.

6 GDE for Sudoku

The Sudoku puzzle is a good candidate to test new algorithmic ideas because it is entertaining and instructive as well as a non-trivial constrained combinatorial problem. We have used it in previous work to test GPSO [16] and a GA [17] with geometric operators based on a vector-of-permutations solution representation, which is a natural representation for Sudoku grids, associated with row-wise swap distance. In this section, we derive the specific GDE for Sudoku based on the space above. Then, in section 6.4, we present experimental results and compare the performance of GA, GPSO and GDE.

6.1 Sudoku solving as optimization problem

Sudoku is a logic-based placement puzzle. The aim of the puzzle is to enter a digit from 1 through 9 in each cell of a 9x9 grid made up of 3x3 subgrids (called "regions"), starting with various digits given in some cells (the "givens"). Each row, column, and region must contain only one instance of each digit. Sudoku puzzles with a unique solution are called proper sudoku, and the majority of published grids are of this type. The general problem of solving Sudoku puzzles on $n^2 \times n^2$ boards of n x n blocks is known to be NP-complete [28].

Sudoku is a constraint satisfaction problem with 4 types of constraints:

1. Fixed elements
2. Rows are permutations
3. Columns are permutations
4. Boxes are permutations

It can be cast as an optimization problem by choosing some of the constraints as hard constraints that all feasible solutions have to respect, and the remaining constraints as soft constraints that can be only partially fulfilled and the level of fulfillment is the fitness of the solution. Only two of the above four constraints can be chosen to be hard constraints because generating feasible initial grids satisfying three or more constraints is NP-Hard [17]. We consider a space with the following characteristics:

- *Hard constraints*: fixed positions and permutations on rows
- *Soft constraints*: permutations on columns and boxes
- *Distance*: sum of swap distances between paired rows (row-swap distance)

Formally, the fitness function (to maximize) is

$$f(g) = \sum_{i=1\ldots9} \sum_{e=1\ldots9} \delta(\sharp(e, row_g(i)) = 1) + \sum_{i=1\ldots9} \sum_{e=1\ldots9} \delta(\sharp(e, col_g(i)) = 1) + \sum_{i=1\ldots9} \sum_{e=1\ldots9} \delta(\sharp(e, box_g(i)) = 1)$$

where $\sharp(e, v)$ is a function that returns the number of occurrences of the number e in the vector v; $\delta(c)$ is a function that returns 1 if the condition c is true, and zero otherwise; $row_g(i)$, $col_g(i)$ and $box_g(i)$ are vectors containing the elements of, respectively, the i-th row, column and box of the grid g. In words, it is the sum of the number of unique elements in each row, plus, the sum of the number of unique elements in each column, plus, the sum of the number of unique elements in each box. So, for a 9×9 grid we have a maximum fitness of $9 \cdot 9 + 9 \cdot 9 + 9 \cdot 9 = 243$ for a completely correct Sudoku grid and a minimum fitness little more than $9 \cdot 1 + 9 \cdot 1 + 9 \cdot 1 = 27$ because for each row, column and square there is at least one unique element type.

It is possible to show that the fitness landscapes associated with this space is smooth, making the search operators proposed a good choice for Sudoku.

In previous work [17], we presented geometric crossovers and mutations based on the space of vectors of permutations endowed with the row-swap distance. The geometric mutation swaps two non-fixed elements in a row. The geometric crossovers are the row-wise PMX (Partially Matched Crossover) and row-wise cycle crossover that recombine parent grids applying respectively PMX and cycle crossover to each pair of corresponding rows. These operators preserve the hard constraints above, hence search only the space of feasible solutions (when the initial population is seeded with feasible solutions).

6.2 Extension ray and Convex combination in product spaces and subspaces

In the following, we present general theoretical results that allow us to build new convex combination (or extension ray recombination) by combining operators that are known to be convex combinations (or extension ray recombinations) and by restricting the domain of known convex combinations (or extension ray recombinations). These results are very useful to deal in a natural way with the compound structure of Sudoku solutions and their hard constraints. We illustrate their application to Sudoku in the following section. Notice that the results on convex combination presented in this section refine those presented earlier within the GPSO framework [16].

Theorem 7. *The operator on the product space obtained by combining vector-wise a set of convex combination operators is a convex combination on the product space endowed with the distance obtained by summing the distances of the composing convex operators.*

Proof. Let us consider the convex combination operators $CX_1(S_1, S_1) \to S_1, CX_2(S_2, S_2) \to S_2, ..., CX_n(S_n, S_n) \to S_n$. Let the compound operator on the product space $S = S_1 \times S_2 \times ... \times S_n$ $CX(S, S) \to S$ be defined as $CX(S, S) =$

$(CX_1(S_1, S_1), CX_2(S_2, S_2), ..., CX_n(S_n, S_n))$. Since $CX_1, CX_2, ..., CX_n$ are convex combination operators they are also geometric crossover under distances $d_1, d_2, ..., d_n$. For the product geometric crossover theorem [14], the compound operator CX is a geometric crossover under the distance $d = d_1 + d_2 + ... + d_n$.

To prove that CX is a convex combination on d, we need to prove that applying $CX_1, CX_2, ..., CX_n$ all with the same parent weights W_a and W_b on their respective spaces $(S_1, d_1), (S_2, d_2), ..., (S_n, d_n)$ and grouping their offspring in a vector is equivalent to applying CX on the space (S, d) with weights W_a and W_b. Let $c' = CX_1(a', b'), c'' = CX_2(a'', b''), ..., c^{(n)} = CX_n(a^{(n)}, b^{(n)})$. We have that $d_1(a', c') = d_1(a', b') \cdot W_b, d_2(a'', c'') = d_2(a'', b'') \cdot W_b, ..., d_n(a^{(n)}, c^{(n)}) = d_n(a^{(n)}, b^{(n)}) \cdot W_b$. Summing these equations we obtain $d_1(a', c') + d_2(a'', c'') + ... + d_n(a^{(n)}, c^{(n)}) = (d_1(a', b') + d_2(a'', b'') + ... + d_n(a^{(n)}, b^{(n)})) \cdot W_b$. This can be rewritten in terms of the distance d as $d((a', a'', ..., a^{(n)}), (c', c'', ..., c^{(n)})) = d((a', a'', ..., a^{(n)}), (b', b'', ..., b^{(n)})) \cdot W_b$. An analogous result holds for the parents $b', b'', ..., b^{(n)}$ with respect to the weight W_a. This means that CX is a weighted combination with respect to the distance d. □

Theorem 8. *The operator on the metric sub space (S', d) with $S' \subset S$ obtained by restricting the domain of application of a convex combination operator on the metric space (S, d) from S to S' is a convex combination operator on (S', d).*

Proof. Let $C = \{c_i\}$ be the set of offspring obtained by $CX(a, b)$ with weights W_a and W_b on the original space (S, d). The operator CX is a convex combination if and only if for any c_i we have that $d(a, c_i) + d(c_i, b) = d(a, b)$ and that $d(a, c_i)/d(c_i, b) = W_b/W_a$. By restricting the space S to $S' \subset S$, we have that if $a, b \in S'$ then the set C' of their offspring by $CX(a, b)$ with weights W_a and W_b on the restricted space is $C' \subset C$. The properties on each of the offspring in C' defining the convex combination operator CX on the subspace S' holds because they hold on for every offspring in the superset C. □

The product space theorem and the sub space theorems apply as well to extension ray operators. Essentially the reason is because the equality $c = CX(a, b)$ involving the convex combination CX with weights W_a, W_b and the equality $b = ER(a, c)$ involving the extension ray recombination ER with weights W_a, W_b, from a declarative point of view are equivalent, as they entail exactly the same relationship between the points a, b and c, which is, the point c is in the line between the points a and b and their distances to it are inversely proportional to their weights, i.e., $d(a, c) \cdot W_a = d(b, c) \cdot W_b$. The two operators differ in which among a, b and c are known elements and which are unknown. In the case of CX, a and b are known and c unknown; in the case of ER, a and c are known and b unknown. Since the theorems above do not rely on this difference, they apply to both CX and ER.

The correct practical application of these theorems may require careful understanding of the difference between declarative definition and operational definition of the recombination operators. Also, these theorems hold when distances are deterministic objects. However, in the operators defined in this paper distances are treated as stochastic objects (random variables) and distance relationship between points are guaranteed only in expectation. Special care needs to be taken when applying these theorems on stochastic operators.

6.3 Convex combination and extension ray recombination for Sudoku

In this section we use the theoretical results in the previous section, to build the convex combination and extension ray recombination operators for Sudoku starting from those for permutations under swap distance. As usual, once we have these specific operators we can plug them in the formal GDE (algorithm 2) to obtain a specific GDE for the space of vectors of permutations under row-wise swap distance, hence obtaining a specific GDE for Sudoku.

The product convex combination theorem allows us to build a convex combination for an entire Sudoku grid by applying row-wise a convex combination operator defined on permutations. Let cx be a convex combination operators on permutations under swap distance (as the one presented in algorithm 6), with weights W_a and W_b, and p_a, p_b, p_c be the two parent permutations and the offspring permutation respectively, i.e., $p_c = cx(p_a, p_b)$. By applying cx to each paired rows of sudoku grids G_a and G_b and grouping the offspring permutations in a grid G_c, we obtain a convex combination operator CX on grids under row-wise swap distance, with weights W_a and W_b.

We want to search the subspace of Sudoku grids in which all givens are fixed. Grids belonging to this subspace are termed feasible grids. We note that when the convex combination operator CX on grids is applied to feasible grids it returns feasible grids. This holds because any convex combination operator on permutations under swap distance transmits unchanged to the offspring those elements that are in the same positions in both parents. Therefore, for the subspace convex combination theorem, the operator CX' obtained by restricting the domain of CX to feasible grids is a convex combination operator on the space of feasible grids (w.r.t. the same metric).

Analogously to the product convex combination theorem, the product extension ray theorem allows us to build an extension ray recombination operator ER on entire Sudoku grids by applying row-wise an extension ray recombination operator defined on permutations (such as the one in algorithm 7).

Analogously to the subspace convex combination theorem, the subspace extension ray theorem can be used to derive an operator ER' from the operator ER to the search the subspace of feasible grids, provided that the ER operator returns feasible grids when applied to feasible grids. This can be achieved by modifying the extension ray operator on permutations in algorithm 7 as a base for ER, in a way that fixed elements in both parents are not changed (so obtaining feasible offspring). However also the probability p for the extension ray recombination operator must be changed and adapted to the new restricted space. In fact, if one prevents the fixed elements to be changed its effect is to have less swaps applied to b to obtain c, which is, a shorter expected swap distance between b and c. To compensate for this is sufficient to increment adequately the probability p of swaps to obtain the wanted expected swap distance between b and c. This probability can be easily determined by noticing that searching a permutation subspace with permutations of size n with ng fixed elements is, in fact, equivalent to search a permutation space with permutations of size $n - np$ obtained by removing the fixed elements that can be added back when the search is over. So the probability p for the extension ray recombination operator on this space is $p = SD(p_b, p_c)/(n - ng - 1 - SD(p_a, p_b))$.

6.4 Experiments with GDE on Sudoku

We implemented GDE for Sudoku in the same publicly available codebase as our previous experiments on evolutionary algorithms with geometric operators and geometric particle swarm optimization [16] [17]. As our previous results define the state of the art for this problem, we chose to compare our GDE results with our previous results, and have thus not run any additional experiments for other algorithms than GDE.

The same parameter search was performed as for Sudoku as for TSP (see section 5.4 for details). However, instead of 20 randomly generated instances the algorithms were tested on two of the same Sudoku grids as used in our previous papers, one rated as "easy" and the other as "hard" by a Sudoku web site. For each parameter configuration, the algorithm was run 50 times. Average fitness was calculated, as well as the number of times out of 50 the run resulted in a fitness of 243, which means the grid was solved.

From Table 5 we can see that GDE is on par with a finely-tuned GA on both easy and hard

Algorithm	Easy 1	Hard 1
Hillclimber	35	1
GA	50 (243)	15 (242)
GPSO	36 (242)	N/A
GDE	50 (243)	13 (242)

Table 5: Number of runs out of 50 the grid was solved (average highest fitness, rounded to the nearest integer, in parentheses); GDE compared with other search algorithms from previous papers, on the same two Sudoku grids. The best results found after parameter tuning are reported for all algorithms. The best GDE settings were population size 50, F 1.0, Cr 0.8 for Easy 1, and population size 100, F 0.0, Cr 0.6 for Hard 1.

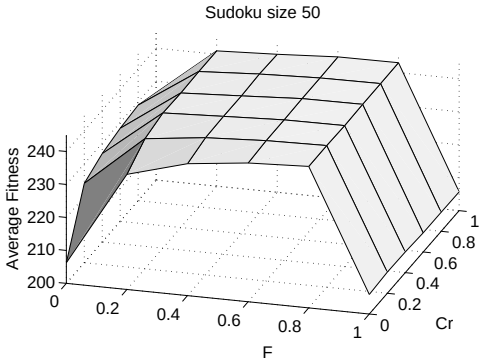

Figure 4: Parameter settings and corresponding average fitness on the "easy 1" Sudoku grid , using population size 50, where the best setting was found.

Sudoku grids, and significantly outperforms both Geometric PSO and hill climbers. It should be noted that more extensive tuning was performed for the GA than for GDE for this problem, as a number of different geometric crossover and mutation operators were tried; similar attention given to the GDE might improve the results further.

Figure 4 presents a comparison of parameter settings for GDE with population size 50 on the easy grid. We can see a general preference for high values of both parameters, though the effect is stronger for Cr than for F. Additionally, extreme values of Cr (0 and 1) yield much lower performance, which is understandable as these lead to a degenerate algorithm. The best parameter setting for the Hard 1 grid yields very good results, though not the best results found, on the Easy 1 grid. In general, it seems that a Cr between 0.4 and 0.8 works well for all tested Sudoku problems, almost regardless of the F value.

6.4.1 Discussion

The good performance of GDE on the Sudoku problem is consonant with the good performance on small instances on TSP; this is because each individual Sudoku grid is a composite of 9 permutations of length 9, slightly smaller than the shortest TSP tours tested. The optimal parameters for Sudoku differ from those for TSP, but the general pattern remains that:

- Both parameters are important, suggesting that both operators contribute to search quality;
- extreme values of Cr yield drastic performance decreases; and
- within the intermediate parameter range $0.2 \leq Cr \leq 0.8$ and $F \geq 0.2$ the algorithm

performs quite well.

7 GDE for Genetic Programs

In order to specify the GDE algorithm to the specific space of genetic programs, we need to choose a distance between genetic programs. A natural choice of distance would be a distance (metric) associated to the Koza-style crossover [7]. This would allow us to derive the specific GDE that searches the same fitness landscape seen by that crossover operator. Unfortunately, the Koza-style crossover is provably non-geometric under any metric [15], so there is no distance associated with it[7] we can use as basis for the GDE. Another crossover operator, the homologous crossover [8] is provably geometric under structural Hamming distance (SHD) [13] which is a variant of the well-known structural distance for genetic programming trees [2]. We use this distance as basis for the GDE because we will be able to use the homologous crossover as a term of reference. Notice, however, that in principle, we could choose any distance between genetic programming trees as a basis of the GDE.

7.1 Homologous crossover and structural Hamming distance

The common region is the largest rooted region where two parent trees have the same topology. In homologous crossover [8] parent trees are aligned at the root and recombined using a crossover mask over the common region. If a node belongs to the boundary of the common region and is a function then the entire sub-tree rooted in that node is swapped with it.

The structural distance [2] is an edit distance specific to genetic programming trees. In this distance, two trees are brought to the same tree structure by adding null nodes to each tree. The cost of changing one node into another can be specified for each pair of nodes or for classes of nodes. Differences near the root have more weight. The structural Hamming distance [13] is a variant of the structural distance in which when two subtrees are not comparable (roots of different arities) they are considered to be at a maximal distance. When two subtrees are comparable their distance is at most 1.

Definition 8. (Structural Hamming distance (SHD))
$dist(T_1, T_2) = hd(p, q)$ if $arity(p) = arity(q) = 0$
$dist(T_1, T_2) = 1$ if $arity(p) \neq arity(q)$
$dist(T_1, T_2) = \frac{1}{m+1}(hd(p, q) + \sum_{i=1\ldots m} dist(s_i, t_i))$ if $arity(p) = arity(q) = m$

Theorem 9. *Homologous crossover is a geometric crossover under SHD [13].*

7.2 Convex combination

In the following, we first define a weighted version of the homologous crossover. Then we show that that operator is a convex combination in the space of genetic programming trees endowed with SHD. In other words, the weighted homologous crossover implements a convex combination CX in this space.

Definition 9. (Weighted homologous crossover). Let P_1 and P_2 be two parent trees, and W_1 and W_2 their weights, respectively. Their offspring O is generated using a crossover mask on the common region of P_1 and P_2 such that for each position of the common region, P_1 nodes appear in the crossover mask with probability W_1, and P_2 nodes appear with probability W_2.

Theorem 10. *The weighted homologous crossover is in expectation a convex combination in the space of genetic programming trees endowed with SHD.*

Proof. The weighted homologous crossover is a special case of homologous crossover so it is also geometric under SHD. Therefore, the offspring of the weighted homologous crossover

[7] In the sense that there is no distance such that the offspring trees are always within the metric segment between parent trees.

are on the segment between parents as required to be a convex combination. To complete the proof we need to show that the weights W_1 and W_2 of the weighted homologous crossover are inversely proportional to the expected distances $E[SHD(P_1, O)]$, $E[SHD(P_2, O)]$ from the parents P_1 and P_2 to their offspring O, i.e., $E[SHD(P_1, O)] \cdot W_1 = E[SHD(P_2, O)] \cdot W_2$, as follows.

Given two trees P_1 and P_2, the SHD can be seen as a weighted Hamming distance on the common region of P_1 and P_2 where the weight W_i on the distance of the contribution of a position i in the common region depends on the arities of the nodes on the path from i to the root node. For each position i of the common region, the expected contribution $SHD_i(P_1, O)$ to the distance $SHD(P_1, O)$ of that specific position is directly proportional to W_i and inversely proportional to the weight W_1 (so, $E[SHD_i(P_1, O)] = W_i/W_1$). This is because, from the definition of weighted homologous crossover, W_1 is the probability that at that position the offspring O equals the parent P_1. So, the higher this probability, the smaller the expected contribution to the distance at that position. Furthermore the contribution to the distance is proportional to the weight W_i of the position i by definition of weighted Hamming distance. From the linearity of the expectation operator, we have that $E[SHD(P_1, O)] = E[\sum_i SHD_i(P_1, O)] = \sum_i E[SHD_i(P_1, O)] = \sum_i W_i/W_1 = 1/W_1$. The last passage holds true because by definition of SHD the sum of the weights on the common region equals 1 (this corresponds to the case of having two trees maximally different on the common region and their distance is 1). Analogously, for the other parent one obtains $E[SHD(P_2, O)] = 1/W_2$. This completes the proof. □

7.3 Extension ray

In the following, we first define a weighted homologous recombination. Then we show that this operator is an extension ray recombination in the space of genetic programming trees endowed with SHD.

To determine a recombination that implements an extension ray operator, it is useful to think of an extension ray operator as the inverse of a convex combination operator, as follows. Given a parent P_1 (the origin of the extension ray) and the offspring C (the point the extension ray passes through), one wants to determine a parent P_2 (the point on the extension ray) such that O results from the convex combination of P_1 and P_2. The weighted extension ray homologous recombination is described in Algorithm 8. It produces offspring with the same tree structure of the second parent.

Notice that in theory any arbitrarily large subtree S_C could be generated to be included in T_C. However, in practice its size should be limited. In the experiment, we generate S_C with the same number of nodes of S_A and S_B.

Theorem 11. *The weighted extension homologous ray recombination is in expectation an extension ray operator in the space of genetic programming trees endowed with SHD.*

Proof. First we prove that $T_C = ER(T_A, T_B)$ by showing that $T_B = CX(T_A, T_C)$. Then we prove that the expected distances $E[SHD(T_A, T_B)]$ and $E[SHD(T_B, T_C)]$ are inversely proportional to the weights W_{AB} and W_{BC}, respectively, i.e., $E[SHD(T_A, T_B)] \cdot W_{AB} = E[SHD(T_B, T_C)] \cdot W_{BC}$.

The offspring T_C has the same structure of T_B. This is because T_C was constructed starting from T_B and then for each node of the common region between T_A and T_B, T_C was not changed or it was randomly chosen but preserving the arity of that node in T_B.

The structures of the common regions $CR(T_A, T_B)$ and $CR(T_A, T_C)$ coincide. This is because the structure of the common region between two trees is only function of their structures. So, since T_B and T_C have the same structure, $CR(T_A, T_B)$ and $CR(T_A, T_C)$ have the same

Algorithm 8 Weighted extension ray homologous recombination

inputs: parent trees T_A (origin point of the ray) and T_B (passing through point of the ray), with corresponding weights W_{AB} and W_{BC} (both weights are between 0 and 1 and sum up to 1)
output: a single offspring tree T_C (a point on the extension ray beyond T_B on the ray originating in T_A and passing through T_B)
compute the structural Hamming distance $SHD(T_A, T_B)$ between T_A and T_B
set $SHD(T_B, T_C) = SHD(T_A, T_B) \cdot W_{AB}/W_{BC}$ (compute the distance between T_B and T_C using the weights)
set $p = SHD(T_B, T_C)/(1 - SHD(T_A, T_B))$ (the probability p of flipping nodes in the common region away from T_A and T_B beyond T_B)
set $T_C = T_B$
for all position i in the common region between T_A and T_B **do**
    consider the paired nodes $T_B(i)$ and $T_A(i)$ in the common region
    if $T_B(i) = T_A(i)$ and random(0,1) $\leq p$ **then**
        set $T_C(i)$ to a random node with the same arity of $T_A(i)$ and $T_B(i)$
    end if
end for
return string T_C as offspring

structure.

The tree T_B can be obtained by homologous crossover applied to T_A and T_C (hence, $T_C = ER(T_A, T_B)$). This can be shown considering two separate cases, (i) nodes of T_B inherited from the common region $CR(T_A, T_C)$ and (ii) subtrees of T_B inherited from subtrees of T_A and T_C at the bottom of the common region. Let us consider nodes on the common region. For each node with index i in the common region, the node $T_B(i)$ matches $T_A(i)$ or $T_C(i)$. This is true from the way $T_C(i)$ was chosen on the basis of the values of $T_A(i)$ and $T_B(i)$. We have two cases. First, $T_C(i)$ was chosen at random, when $T_A(i) = T_B(i)$. In this case $T_B(i)$ can be inherited from $T_A(i)$, since it may be $T_B(i) \neq T_C(i)$ but $T_B(i) = T_A(i)$. Second, $T_C(i)$ was chosen to equal $T_B(i)$, when $T_A(i) \neq T_B(i)$. In this case $T_B(i)$ can be inherited from $T_C(i)$. In either case, for nodes on the common region the corresponding nodes of T_B can be inherited from T_A or T_C. The subtrees of T_B at the bottom of the common region can be inherited all from T_C (both structures and contents), since by construction T_C inherited those subtrees from T_B without modifying them.

To show that this recombination is a weighted extension homologous ray recombination, we are left to show that the expected distances $E[SHD(T_A, T_B)]$ and $E[SHD(T_B, T_C)]$ are inversely proportional to the weights W_{AB} and W_{BC}, i.e., $E[SHD(T_A, T_B)] \cdot W_{AB} = E[SHD(T_B, T_C)] \cdot W_{BC}$. The probability p of flipping nodes in the common region away from T_A and T_B beyond T_B was chosen as an appropriate function of W_{AB} and W_{BC} and of $SHD(T_A, T_B)$ to obtain $SHD(T_B, T_C)$ such that the above requirement holds true. It is possible to prove that the chosen p is the correct one using the same argument used in the proof of theorem 10.

□

Now we have operational definitions of convex combination and extension ray for the space of genetic programming trees under SHD. These space-specific operators can be plugged in the formal GDE (algorithm 2) to obtain a specific GDE for the genetic programming trees space, the GDE-GP.

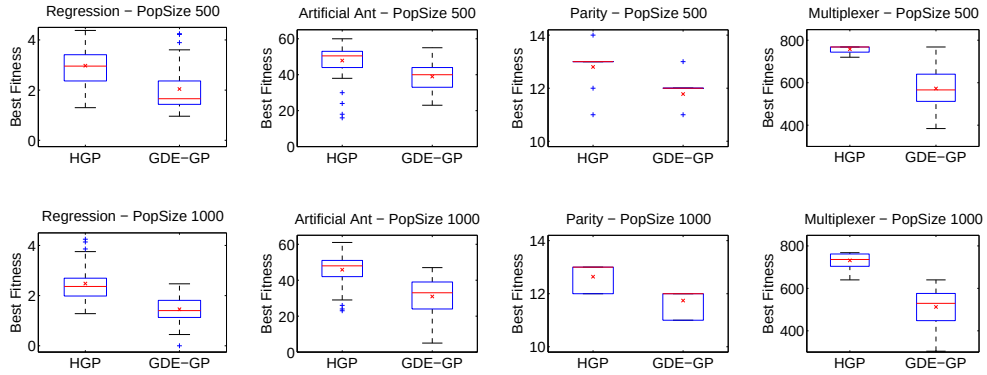

Figure 5: Boxplots of the best fitness achieved in each problem ($\times$ marks the mean). Population sizes of 500 individuals (top row) and 1000 individuals (bottom row)

7.4 Experiments for GDE-GP

This section reports an experimental analysis of the GDE-GP behavior on four standard GP benchmark problems: Symbolic Regression of the quartic polynomial, Artificial Ant on the Santa Fe trail, 5-Bit Even Parity, and 11-Bit Multiplexer. We will compare the behavior of GDE-GP with a baseline GP approach, not only in terms of fitness but also in terms of the evolution of population diversity and average solution length. We also take a closer look at pairwise and average SHDs during the evolution in order to understand the different behaviors.

In all the experiments we used $F = 0.8$ and $Cr = 0.9$, according to [23]. As a baseline for comparison we used standard GP with homologous crossover (70%) and reproduction (30%), always applying point mutation with probability $1/L$, where L is the number of nodes of the individual. We call this baseline HGP. All the experiments were performed using populations of two different sizes (500 and 1000 individuals) initialized with the Ramped Half-and-Half procedure [7] with an initial maximum depth of 8, allowed to evolve for 50 generations. Each experiment was repeated 50 times. Statistical significance of the null hypothesis of no difference was determined with pairwise Kruskal-Wallis non-parametric ANOVAs at $p = 0.05$. A non-parametric ANOVA was used because the data is not guaranteed to follow a normal distribution. For the same reason, the median was preferred over the mean in all the evolution plots that follow. The median is also more robust to outliers.

Figure 5 shows the boxplots of the best fitness achieved along the run, using populations of 500 individuals (top row) and 1000 individuals (bottom row). With a population size of 500, in all four problems there is a statistically significant difference between HGP and GDE-GP. GDE-GP is consistently better than HGP.

It may be argued that HGP is being crippled by such a small population size, which may reduce diversity along the run. This could be true, because when doubling the population size HGP significantly improves its best fitness of run in both Regression and Multiplexer problems. However, with 1000 individuals the GDE-GP techniques show significant improvements in many more cases, and remain consistently better than HGP, exactly as before.

However, the observation of diversity, measured as the percentage of genotypically distinct individuals in the population, revealed somewhat unexpected results. Figure 6 shows the evolution of the median values of diversity along the run, for both population sizes. Not only does not HGP show any clear signs of diversity loss, regardless of population size, but GDE-GP exhibits an extraordinarily varied behavior, approaching both extreme values in different problems

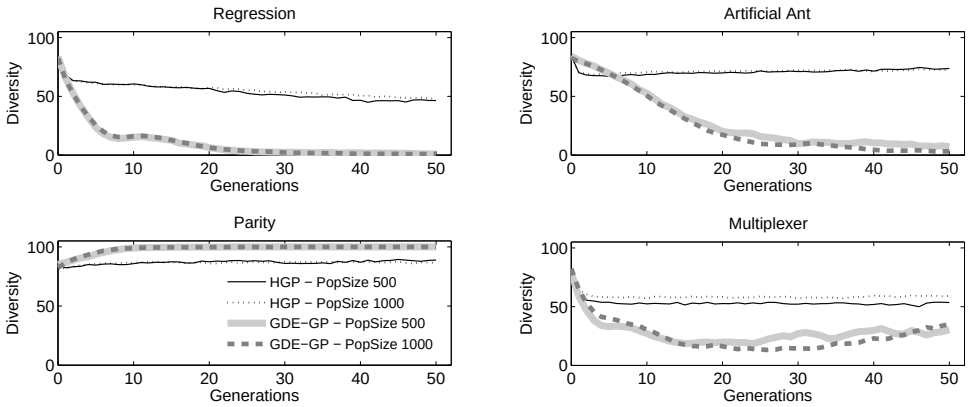

Figure 6: Evolution of the median values of diversity in each problem

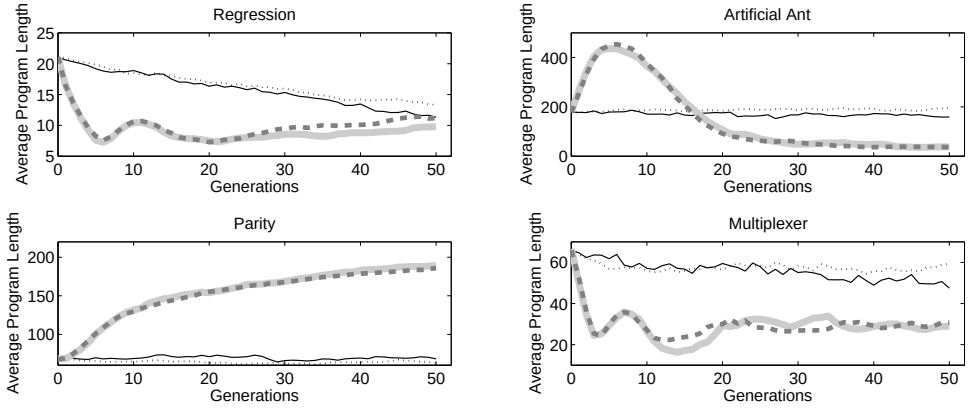

Figure 7: Evolution of the median values of average program length in each problem. Legend as in Figure 6

(in Regression and Artificial Ant it practically reaches 0% while in Parity it reaches 100%), in some cases undergoing large fluctuations along the run (Multiplexer).

In Figure 7 we look at the evolution of the median values of average program length along the run, for both population sizes. Once again GDE-GP behaves radically differently from HGP. GDE-GP presents large but smooth fluctuations in most problems, when compared to the more constrained but somewhat erratic behavior of HGP. The most interesting case is probably the Artificial Ant, where GDE-GP quickly and steadily increases the average program length until a plateau is reached, followed by a steep decrease to very low values and a tendency for stabilization at the end of the run.

In spite of the large fluctuations in diversity and average program length observed in the GDE-GP runs, nothing particularly obvious seems to simultaneously happen in terms of fitness evolution, except for the fact that fitness seems to stabilize when either diversity or average program length reach very low values (not shown). There is also no clear correlation between the changes in diversity and the changes in average program length.

In an attempt to understand the reasons behind the different evolution patterns of GDE-GP and HGP we have observed what happens in terms of the distance between the individuals during

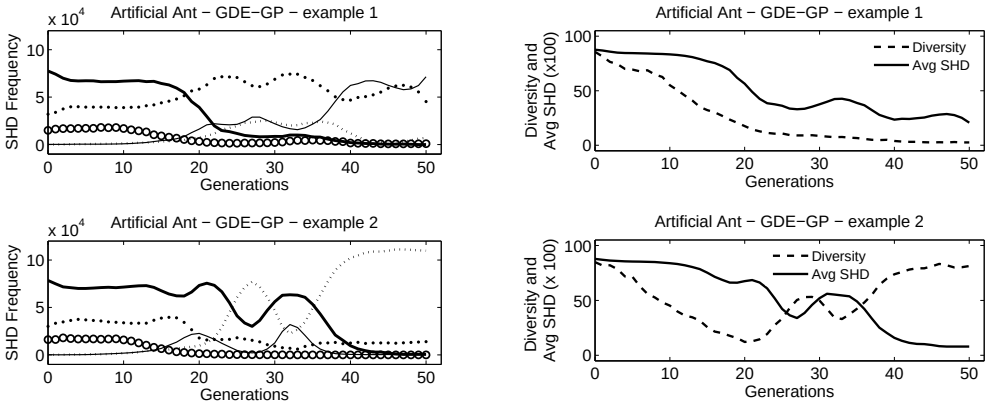

Figure 8: Two GDE1 runs of Artificial Ant with population size 500, showing the evolution of SHD frequencies (left) and the evolution of diversity and average SHD (right). Legend for the frequency plots: 0]0,0.25[[0.25,0.75[[0.75,1[1

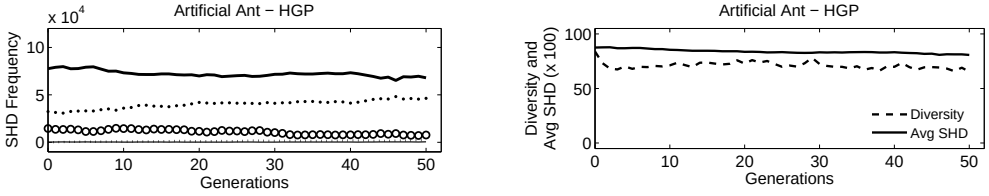

Figure 9: HGP run of Artificial Ant with population size 500, showing the evolution of SHD frequencies (left) and the evolution of diversity and average SHD (right). Legend as in Figure 8

the search process, and tried to visualize how the individuals "move" in the search space. In each generation we measured all the pairwise SHDs between the trees of the population. Figure 8 reports two examples of Artificial Ant runs performed with population size 500. The plots on the left show the evolution of the frequency of pairs within each of five different distance intervals. The intervals considered are: $[0, 0]$ (minimal distance, equal to 0), $]0, 0.25[$ (short distances), $[0.25, 0.75[$ (medium distances), $[0.75, 1[$ (long distances), and $[1, 1]$ (maximal distance, equal to 1). The plots on the right show the evolution of diversity along with the evolution of the average SHD (multiplied by 100, so it uses the same range as diversity). The two runs reported on this figure are interesting because they show two possible and quite opposite behaviors in terms of SHD evolution. In the first one there is a clear predominance of medium distances from the middle of the run, shared with an even greater predominance of minimal distances at the end of the run, which accounts for the loss of diversity as well as the low average SHD. In the second example the run finishes with an almost complete dominance of the short distances, evidencing a strong convergence, but with no minimal distances, which accounts for a high diversity in spite of the low average SHD. Because the maximal distances take longer to disappear in this example, for many generations it is the maximal and the short distances that dominate, which strongly suggests the formation of clusters of individuals in the search space (merging into only one cluster by the end of the run).

Figure 9 reports one typical example of an Artificial Ant run performed with HGP with population size 500. Although the initial frequencies of distances are the same as in GDE-GP,

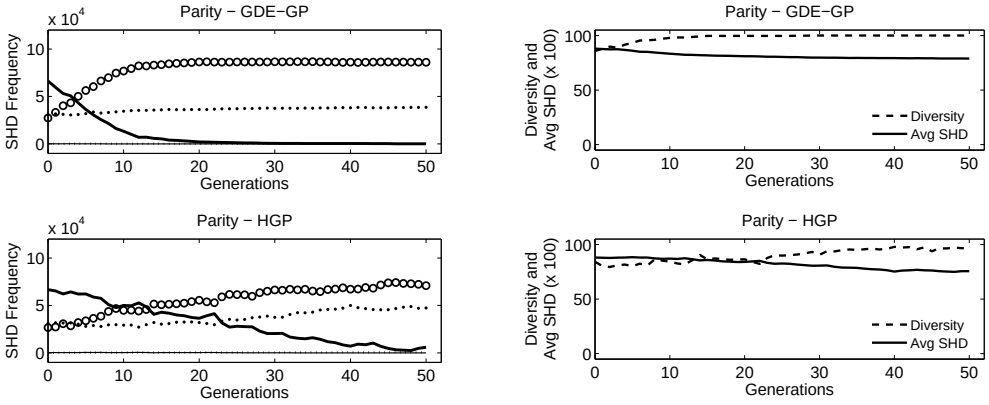

Figure 10: GDE1 and HGP runs of Parity with population size 500, showing the evolution of SHD frequencies (left) and the evolution of diversity and average SHD (right). Legend as in Figure 8

their evolution is radically different in HGP.

Unlike Artificial Ant, Parity was the problem where the behavior of GDE-GP revealed almost no variability between different runs, and also revealed to be very similar to the behavior of HGP in terms of diversity (Figure 6). Similarly to the previous figures, Figure 10 reports examples of the typical behavior of GDE-GP (top) and HGP (bottom) in the Parity problem, in terms of the evolution of distance frequencies (left) and average distance along with diversity (right). As expected, the behavior of GDE-GP is different from the one observed in the Artificial Ant examples (Figure 8). Also as expected given the similarities in diversity evolution between GDE-GP and HGP, the behaviors of the two approaches are not so dissimilar as in the Artificial Ant problem.

8 Conclusions

Geometric differential evolution is a formal generalization of DE on continuous spaces that retains the original geometric interpretation and that applies to generic combinatorial spaces. GDE can be formally specified to specific spaces associated, in principle, to any solution representation. In this article, we have illustrated that this is indeed possible in practice by deriving the specific GDEs for the Hamming space associated with binary strings, for the space of permutations endowed with the swap distance, for the space of vectors of permutations endowed with the row-swap distance, and for the space of genetic programs endowed with the structural Hamming distance. These are quite different spaces based on non-trivial solution representations. The derived representation-specific GDEs are, *in a strong mathematical sense*, the same algorithm doing the same type of search on different spaces.

We have analyzed the behavior of specific GDE algorithms experimentally, tested them on standard benchmarks and compared them against a set of classic evolutionary algorithms defined on the same search space and representation. The binary GDE and the GDE-GP outperformed the other algorithms in the comparison. The GDE based on permutations did well on the travelling salesman problem, but only for short tour lengths; for long tours, it performed very badly. Finally, GDE on vectors of permutations on Sudoku did almost as well as a very finely-tuned GA. We believe these are very promising initial results. This is a rather interesting achievement, as the present work is one of the very rare examples in which theory has been able

to inform the practice of search operator design successfully. GDE is a very recent algorithm and further analysis is required to gain an in-depth understanding of its behavior on different representations. Also, further experimentation is needed to explore more thoroughly its potential to solve effectively combinatorial problems and problems naturally formulated as search in spaces of programs. This constitutes an important piece of future work.

The formal generalization methodology employed to generalize differential evolution, which is the same that was used to generalize particle swarm optimization, can be applied in principle to generalize virtually any search algorithm for continuous optimization to combinatorial spaces. Interestingly, this generalization methodology is rigorous, conceptually simple and promising as both GDE and GPSO seem to be quite good algorithms in practice. In future work, we will generalize using this methodology other classical derivation-free methods for continuous optimization that make use of geometric constructions of points to determine the next candidate solution (e.g. Nelder and Mead method and Controlled Random Search method).

Acknowledgements

We would like to thank Riccardo Poli for passing us the code of the homologous crossover for genetic programs, and Paulo Fonseca and João Carriço for ideas on visualizing distances. The first author acknowledges EPSRC grant EP/I010297/1 that partially supported this work. The third author acknowledges project PTDC/EIA-CCO/103363/2008 from FCT, Portugal, and the PIDDAC Program funds of FCT (INESC-ID multiannual funding).

References

[1] A. Caprara, *Sorting by reversals is difficult*, Proceedings of the 1st Annual International Conference on Computational Molecular Biology, 1997, pp. 75–83.

[2] A. Ekart and S. Z. Nemeth, *A metric for genetic programs and fitness sharing*, Genetic Programming, Proceedings of EuroGP'2000, 2000, pp. 259–270.

[3] T. Gong and A. L. Tuson, *Differential evolution for binary encoding*, Soft Computing in Industrial Applications, Springer, 2007, pp. 251–262.

[4] S. Kauffman, *Origins of order: self-organization and selection in evolution*, Oxford University Press, 1993.

[5] J. Kennedy and R. C. Eberhart, *A discrete binary version of the particle swarm algorithm*, IEEE International Conference on Systems, Man, and Cybernetics, 'Computational Cybernetics and Simulation', 1997, pp. 4104–4108.

[6] ———, *Swarm intelligence*, Morgan Kaufmann, 2001.

[7] John R. Koza, *Genetic programming: On the programming of computers by means of natural selection*, The MIT Press, 1992.

[8] W. Langdon and R. Poli, *Foundations of genetic programming*, Springer-Verlag, 2002.

[9] A. Moraglio, *Towards a geometric unification of evolutionary algorithms*, Ph.D. thesis, University of Essex, 2007.

[10] A. Moraglio, C. Di Chio, and R. Poli, *Geometric particle swarm optimization*, European Conference on Genetic Programming, 2007, pp. 125–136.

[11] A. Moraglio, C. Di Chio, J. Togelius, and R. Poli, *Geometric particle swarm optimization*, Journal of Artificial Evolution and Applications **2008** (2008), Article ID 143624, 14 pages.

[12] A. Moraglio and R. Poli, *Topological interpretation of crossover*, Proceedings of the Genetic and Evolutionary Computation Conference, 2004, pp. 1377–1388.

[13] ———, *Geometric landscape of homologous crossover for syntactic trees*, Proceedings of IEEE congress on evolutionary computation, 2005, pp. 427–434.

[14] ———, *Product geometric crossover*, Proceedings of Parallel Problem Solving from Nature conference, 2006, pp. 1018–1027.

[15] ———, *Inbreeding properties of geometric crossover and non-geometric recombinations*, Proceedings of the workshop on the Foundations of Genetic Algorithms, 2007, pp. 1–14.

[16] A. Moraglio and J. Togelius, *Geometric pso for the sudoku puzzle*, Proceedings of the Genetic and Evolutionary Computation Conference, 2007, pp. 118–125.

[17] A. Moraglio, J. Togelius, and S. Lucas, *Product geometric crossover and the sudoku puzzle*, Proceedings of IEEE congress on evolutionary computation, 2006, pp. 470–476.

[18] Alberto Moraglio and Riccardo Poli, *Topological crossover for the permutation representation*, Intelligenza Artificiale **5(1)** (2011), 49–70.

[19] Alberto Moraglio and Julian Togelius, *Geometric differential evolution*, Proceedings of the 11th Annual conference on Genetic and evolutionary computation, 2009, pp. 1705–1712.

[20] Michael O'Neill and Anthony Brabazon, *Grammatical differential evolution*, Proceedings of the 2006 International Conference on Artificial Intelligence, CSREA Press, 2006, pp. 231–236.

[21] G. C. Onwubolu and D. Davendra (eds.), *Differential evolution: A handbook for global permutation-based combinatorial optimization*, Springer, 2009.

[22] G. Pampara, A.P. Engelbrecht, and N. Franken, *Binary differential evolution*, IEEE Congress on Evolutionary Computation, 2006, pp. 1873–1879.

[23] K. V. Price, R. M. Storm, and J. A. Lampinen, *Differential evolution: A practical approach to global optimization*, Springer, 2005.

[24] Rainer Storn and Kenneth Price, *Differential evolution a simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization **11(4)** (1997), 341 – 359.

[25] G. Syswerda, *Uniform crossover in genetic algorithms*, Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 2–9.

[26] Julian Togelius, Renzo De Nardi, and Alberto Moraglio, *Geometric pso + gp = particle swarm programming*, Proceedings of the Congress on Evolutionary Comptutation (CEC), 2008, pp. 3594–3600.

[27] D. Whitley, T. Starkweather, and D. Shaner, *Travelling salesman and sequence scheduling: quality solutions using genetic edge recombination*, Handbook of Genetic Algorithms (L. Davis, ed.), Van Nostrand Reinhold, 1991, pp. 350–372.

[28] T. Yato, *Complexity and completeness of finding another solution and its application to puzzles*, Master's thesis, University of Tokyo, Japan, 2003.